



# Propiedades y Ciclo de vida



Miguel Angel Alvarez  
@midesweb



**Miguel Angel Alvarez**

**miguel@desarrolloweb.com**

**@midesweb**

# En esta sesión:

- Propiedades, converters, accessors
- Observed attributes y reflected attributes
- Ciclo de vida de los componentes
- Métodos de ciclo de vida nativos & LitElement
- Updates asíncronos!



Miguel Angel Alvarez  
@midesweb



# PROPIEDADES y atributos



Miguel Angel Alvarez  
@midesweb

# Propiedades ↔ atributos

LitElement hace el trabajo de sincronizar propiedades del componente y atributos de la etiqueta HTML "host".

Al indicar el "type" de una propiedad, estamos marcando cómo se realiza esa conversión del tipo (pues en atributos sólo se pueden escribir cadenas)

```
static get properties() {  
  return {  
    label: { type: String },  
    placeholder: { type: String },  
    disabled: { type: Boolean },  
    value: { type: String }  
  };  
}
```



Miguel Angel Alvarez  
@midesweb

# Conversores personalizados

Puedes crear conversores que funcionen de maneras diferentes a los que ya ofrece LitElement.

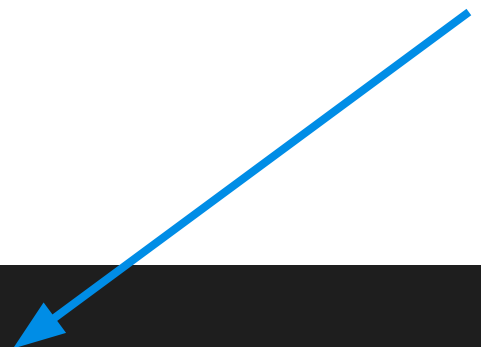
```
value: {  
  converter: {  
    toAttribute(value) {  
      return Math.round(value * 10) / 10;  
    },  
    fromAttribute(value) {  
      return value.toUpperCase();  
    }  
  }  
}
```



Miguel Angel Alvarez  
@midesweb

# Propiedades "reflect"

Indican si se quiere, o no, que los cambios en las propiedades se reflejen en la etiqueta HTML host del componente.



```
value: {  
  reflect: true,  
  converter: {  
    toAttribute(value) {  
      return Math.round(value * 10) / 10;  
    },  
    fromAttribute(value) {  
      return value.toUpperCase();  
    }  
  }  
}
```



Miguel Angel Alvarez  
@midesweb

# Observed attributes

Cada propiedad declarada en un componente se asocia a un atributo. El nombre del atributo es automático (la propiedad en minúsculas), pero lo puedes cambiar.

```
placeholder: {  
  type: String,  
  attribute: 'ph'  
},
```

No tiene nada que ver con los observers de Polymer.  
En LitElement te tienes que acostumbrar a vivir sin observers.



Miguel Angel Alvarez  
@midesweb



# Property accessors

Los accessors permiten establecer acciones o transformaciones cuando se modifica una propiedad o se accede a su valor.

Si creas tus propios accessors tienes que asegurarte de llamar a `requestUpdate()`, pasando el nombre de la propiedad y el valor anterior.

```
static get properties() {  
  return { prop: { type: Number } };  
}  
  
set prop(val) {  
  let oldVal = this._prop;  
  this._prop = Math.floor(val);  
  this.requestUpdate('prop', oldVal);  
}  
  
get prop() { return this._prop; }
```



Miguel Angel Alvarez  
@midesweb



# CICLO DE VIDA de los componentes



Miguel Angel Alvarez  
@midesweb

# Ciclo de vida

Son los momentos por los que va pasando un componente, a medida que se crea y se usa dentro de una página o aplicación web.

Básicamente son métodos que podemos implementar para realizar acciones en momentos del ciclo de vida de los componentes.

- Derivados del estándar Javascript
- Incorporados por LitElement



Miguel Angel Alvarez  
@midesweb

# Ciclo de vida estándar

El estándar de Web Components incluye algunos métodos del ciclo de vida que podemos usar.

- **constructor:** Resume las tareas de inicialización de un componente
- **connectedCallback:** invocado al añadir un elemento al DOM
- **disconnectedCallback:** invocado al retirar un elemento del DOM
- **attributeChangedCallback:** Se produce cuando un atributo cambia
- **adoptedCallback:** cuando un documento se mueve a un nuevo documento.



Miguel Angel Alvarez  
@midesweb



# UPDATES asíncronos



Miguel Angel Alvarez  
@midesweb

# Updates asíncronos

LitElement hace un update asíncrono de los templates cuando cambian sus propiedades. Esto aumenta el rendimiento, puesto que podrían acumularse varios cambios en propiedades que se actualizan de manera conjunta.

## Es importante tenerlo en cuenta!

Puede ocurrir que una propiedad bindeada a otro componente tarde instantes en propagarse. Generalmente son milisegundos, imperceptible para el usuario, pero a nivel de desarrollo puede afectarnos.



Miguel Angel Alvarez  
@midesweb

# Ciclo de updates

A grandes rasgos se realizan los siguientes pasos.

- Una propiedad es modificada
- Comprobar si ese cambio implica un update. En caso positivo, se requiere uno.
- Realizar el update
  - Procesar propiedades y atributos
  - Renderizar el elemento
- Se resuelve una promesa indicando que el update se ha realizado



Miguel Angel Alvarez  
@midesweb

# updateComplete

Devuelve una promesa que se resolverá cuando se acabe de realizar la actualización del template.

```
nameChange() {  
  this.name = Math.random();  
  console.log('value Actual', this.elinput.getValue());  
  this.updateComplete.then(() => {  
    console.log('value después de la promesa', this.elinput.getValue());  
  })  
}
```



Miguel Angel Alvarez  
@midesweb



# requestUpdate

Sirve para solicitar manualmente un update en un template.

```
addRandomTag() {  
  console.log('addRandomTag')  
  let valor = Math.floor(Math.random() * 1000);  
  this.tags.push(valor);  
  this.requestUpdate();  
}
```

Push sobre un array no desencadena la actualización del template

Se puede invocar a requestUpdate() para solicitar "manualmente" la actualización del template.



Miguel Angel Alvarez  
@midesweb



# OTROS EJEMPLOS prácticos



Miguel Angel Alvarez  
@midesweb

# firstUpdated

Se ejecuta después de la primera renderización del componente.

En este momento el template ya está renderizado, por tanto el DOM del componente (shadow dom) ya existe y podemos hacer cosas con él.

"Cacheando" la referencia a objetos del DOM

```
firstUpdated() {  
  this.element = this.shadowRoot.getElementById('textField');  
  this.element.focus();  
}
```

Realizando operaciones sobre los  
elementos del DOM



Miguel Angel Alvarez  
@midesweb

# hasChanged

Es una función configurable en las propiedades declaradas.

Se ejecuta cada vez que cambia una propiedad y sirve para informar a LitElement sobre la conveniencia o no de hacer un update.

=> true (entonces update)

=> false (entonces no hacer update)

```
static get properties() {  
  return {  
    name: {  
      type: String,  
      hasChanged(newVal, oldVal) {  
        console.log(newVal, oldVal);  
        if(newVal * 10 > 5) {  
          return true  
        }  
        return false;  
      }  
    }  
  };  
}
```




Miguel Angel Alvarez  
@midesweb

# updated

Se ejecuta cada vez que el template se ha actualizado. Puedes recibir un listado de las propiedades que han cambiando en el último update.

Esto es un objeto "map" de Javascript

[https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos\\_globales/Map](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Map)



```
updated(changedProperties) {  
  changedProperties.forEach((oldValue, propName) => {  
    console.log(`${propName} changed. oldValue: ${oldValue}`);  
  });  
  this.shadowRoot.getElementById('boton').focus();  
}
```



Miguel Angel Alvarez  
@midesweb



# GRACIAS!!



Miguel Angel Alvarez  
@midesweb